[ADR] Central EventBus

Context

Since `menu` type plugins are siblings in the DOMTree, it is not possible for `menu` type plugins to communicate with each other. Next to the `menu` type plugin, it is not possible to communicate from an `editor` type plugin to another `editor` type plugin or a `menu` type plugin.

Example use-cases

An end-user selects 2 IED's in 1 plug-in. In order to make a connection between IED's, the user want to open the selection in an another plug-in.

Based on specific edits, a vendor specific plug-in can add vendor specific functionality (e.g. Siemens)

XSD validation errors

Also, since addons are not extending from a HTMLElement, addons cannot subscribe to events themselves. HTML events are only going upwards in the DOMTree, not downwards and/or to siblings.

Decision

If we want plug-ins for OpenSCD to be asynchronous accessible to each other, OpenSCD-Core should make use of a central event-bus.
Instead of dispatching an event on the plug-in itself, the plug-in can dispatch an event on the central event-bus.

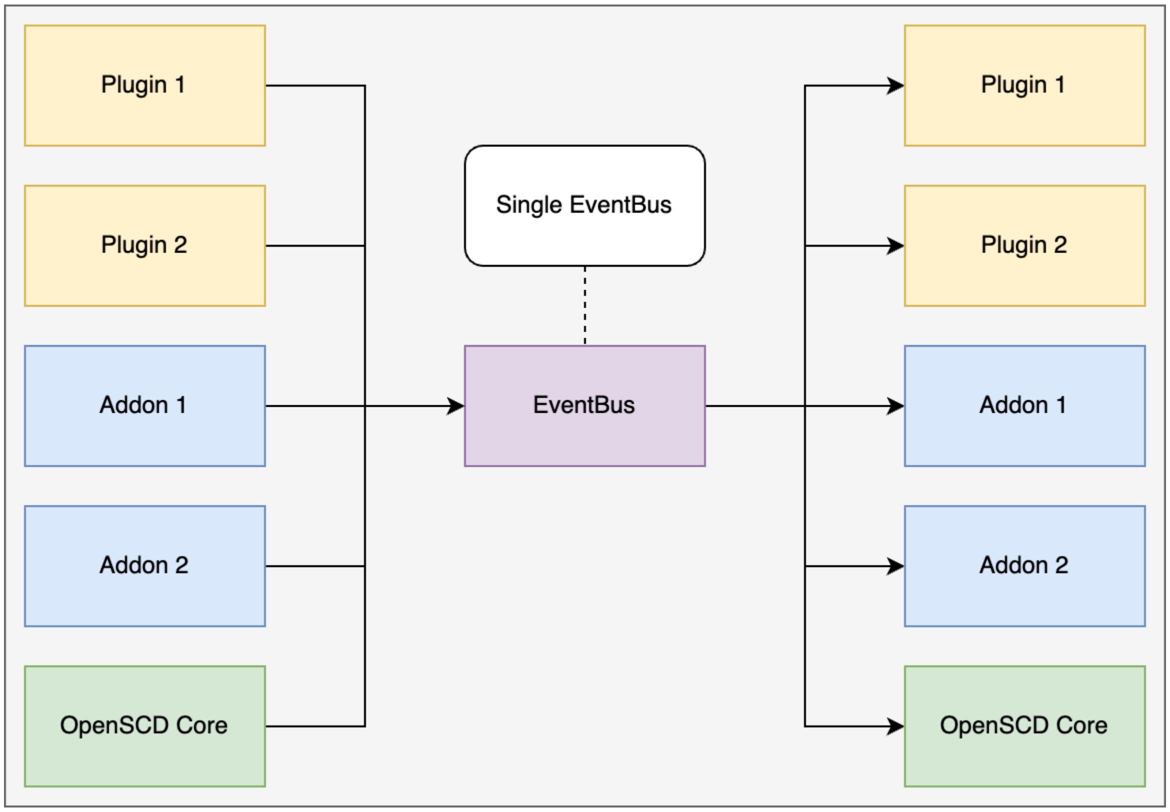
Addons

Addons can listen to CustomEvents being caught by the OpenSCD HTML Element. This will solve the communication problem from a plug-in or addon to a different addon.

Plug-ins communicating between themselves are not solved by this solution.

EventBus implementation

```
export  class EventBus implements EventTarget {

    private  _eventListeners: Map<string, EventListenerOrEventListenerObject[]>;

    constructor() {
        this._eventListeners = new  Map<string, EventListenerOrEventListenerObject[]>();
    }

    addEventListener(type: string, callback: EventListenerOrEventListenerObject): void {
        if (!this._eventListeners.has(type)) {
            this._eventListeners.set(type, []);
        }

        this._eventListeners.get(type)!.push(callback);
    }

    dispatchEvent(event: Event): boolean {
        if (this._eventListeners.has(event.type)) {
            this._eventListeners
                .get(event.type)
                !.forEach((cb) =>
                    typeof  cb === 'function' ? cb(event) : cb.handleEvent(event)
                );

            return  true;
        }

        return  false;
    }

    removeEventListener(type: string, callback: EventListenerOrEventListenerObject): void {
        if (this._eventListeners.has(type)) {
            this._eventListeners.set(type, this._eventListeners.get(type)!.filter((cb) =>  cb !== callback));
        }
    }

}
```



Alternatives

Instead of the central EventBus, we can let plugins dispatch events on the plugin itself. If a plugin needs an event from another plugin, It can listen to an event being dispatched to `Open-SCD Core` .

Example

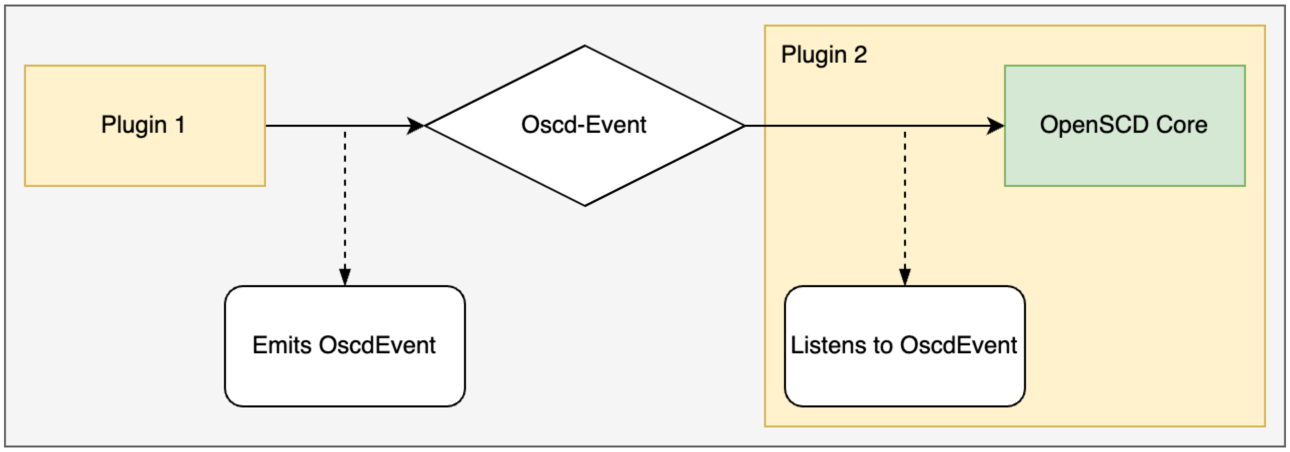
Listening to Events dispatched to OpenSCD Core

```
export default class MyPlugin extends LitElement {
    fistUpdated() {
        this.getRootNode().addEventListener('oscd-event', () => { ... });
    }
}
```

In the code example above, the plugin listens to an OscdEvent dispatched by (for example) another plugin and is listening to this event via the common parent; Open-SCD Core.

- Body
  - OpenSCD
    - Menu Plugin 1
    - Menu Plugin 2
    - Active Editor Plugin

Due to WebApi restrictions, Events that are emitted by one of the plugins are only bubbling up directly (https://www.freecodecamp.org/news/event-bubbling-in-javascript/)



Pro's:

No new Code needs to be introduced

Con's:

This is only working for HTMLElements (Plugins).

It is possible to stop the event from bubbling further inside a plugin. This means that a plugin can negate an event that should be handled by a different plugin. (Plugin A sends an event (that should be read by Plugin B), Plugin C is listening to this event and stops the event from going further (event. preventDefault()). Because of Plugin C, plugin B never gets the event).

Alternative: Window object

Listen to the window object. This object is used everywhere in the OpenSCD application.

Pro's:

Con's

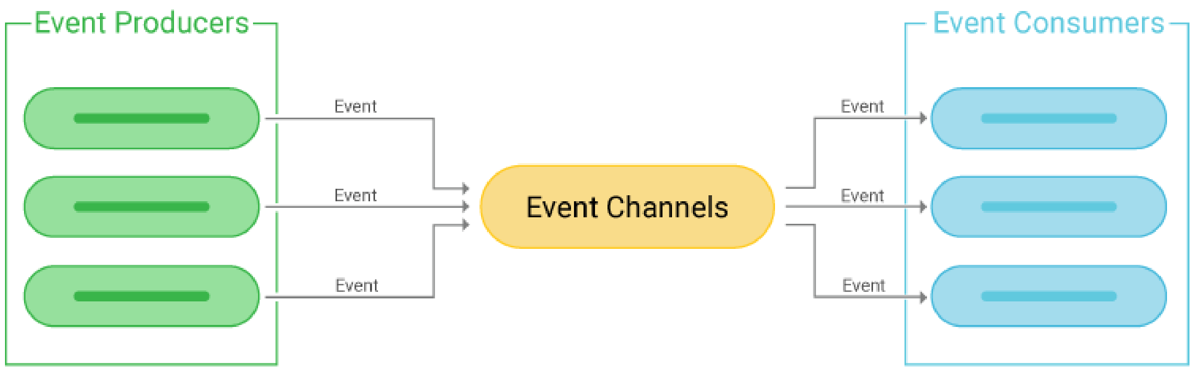
- Risk of becomming a distributored monolith: Distributed Monoliths vs. Microservices: Which Are You Building? | Scout APM Blog
- OpenSCD cannot run in an iframe

Consequences

Implementing a central event-bus means that current OSCD plugins need some refactoring to dispatch events to the event-bus instead of dispatching events on itself.

The semantic on the central procesbus are determined by the producer/consumer.

We can distinguish different type of events by Producer driven or Consumer driven.



Source: https://www.scylladb.com/glossary/event-driven-architecture/

Source: https://www.scylladb.com/glossary/event-driven-architecture/

For example, the `oscd-open` event is a Consumer driven event. The API of this event is established in OpenSCD Core and should be adhered by any plugin.

Producer driven events can be seen as notification events. The producer is notifying anyone who is interested in this event.

Code changes

Current solution

```
class MyPlugin extends HTMLElement {

    onClick() {
        this.dispatchEvent(new Event('oscd-click'));
    }
}
```

EventBus Solution

```
class MyPlugin extends HTMLElement {

    eventBus: EventBus

    onClick() {
        this.eventBus.dispatchEvent(new Event('oscd-click'))
    }
}
```